PATENT ABSTRACTS OF JAPAN

(11)Publication number : 06-222928
(43)Date of publication of application : 12.08.1994

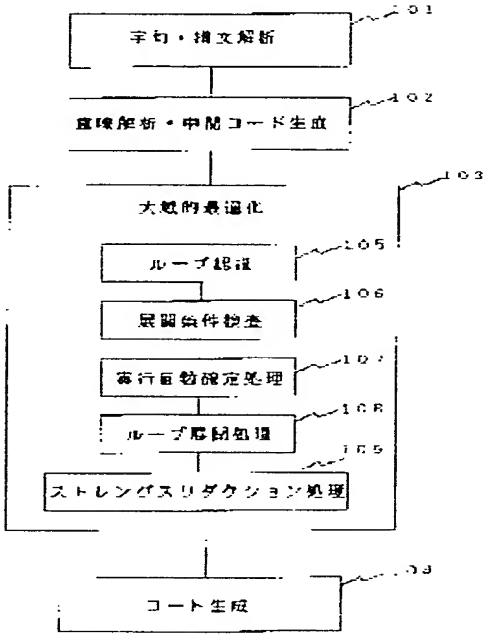
(51)Int.Cl. G06F 9/45

(21)Application number : 05-011369
(22)Date of filing : 27.01.1993

(71)Applicant : HITACHI LTD
HITACHI MICOM SYST:KK
(72)Inventor : TOYAMA KEISUKE
HASHIMOTO AKIRA

(54) LOOP OPTIMIZING METHOD

(57)Abstract:
PURPOSE: To provide a system for improving execution performance by improving the possibility of loop extension and further for executing valid strength reduction even at an RISC computer, for which the use of an index mode is limited, as well, by utilizing that extension concerning the optimizing method for a program.
CONSTITUTION: Concerning a loop, an executing time fixing processing circuit 107 confirms that the number of times of repetition is not changed inside the loop. A loop extension processing part 108 decides the number of times of execution at the time of execution and generates a correspondent extended part together with a code to be branched. A strength reduction processing part 109 further optimizes the extended loop. Thus, even the loop, which can not decide the number of times of repetition at the time of compile, can be extended as well and the optimizing effect of loop extension can be provided. Further, strength reduction can be executed to the main body of the extended loop as well and even in the case of a limited addressing mode observed in a lot of RISC computers, arithmetic is decreased.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号

特開平6－222928

(43)公開日　平成６年(1994)８月12日

(51)Int.Cl.[5]	識別記号	庁内整理番号	ＦＩ	技術表示箇所
Ｇ０６Ｆ　9/45				
		9292－5B	Ｇ０６Ｆ　9/44　322　Ｇ	

審査請求　未請求　請求項の数７　ＯＬ　（全 10 頁）

(21)出願番号　特願平5－11369

(22)出願日　平成５年(1993)１月27日

(71)出願人　000005108
株式会社日立製作所
東京都千代田区神田駿河台四丁目６番地

(71)出願人　000233169
株式会社日立マイコンシステム
東京都小平市上水本町５丁目22番１号

(72)発明者　十山　圭介
神奈川県川崎市麻生区王禅寺1099番地　株式会社日立製作所システム開発研究所内

(72)発明者　橋本　明
東京都小平市上水本町５丁目22番１号　株式会社日立マイコンシステム内

(74)代理人　弁理士　小川　勝男

(54)【発明の名称】　ループ最適化方法

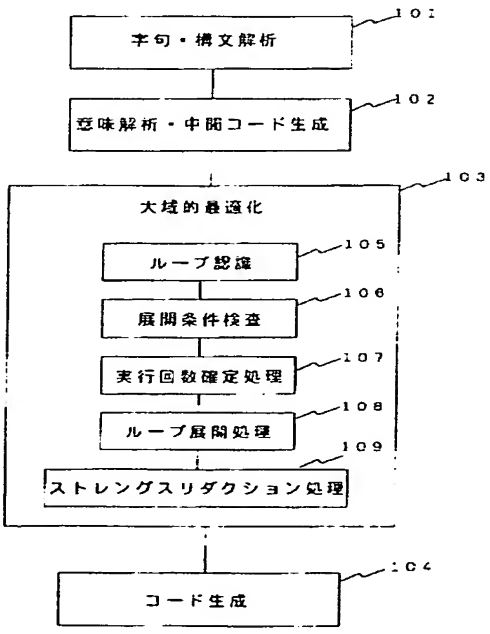
(57)【要約】

【目的】　本発明の目的は、プログラムの最適化手法において、ループ展開の可能性を高めて実行性能を向上させ、さらにその展開を利用して、インテクスモードの使用に制限のあるＲＩＳＣ計算機においても有効なストレングスリダクションを実施する方式を提供することにある。

【構成】　ループに対し、実行回数確定処理部（１０６）で繰返し回数がループ内で変更されないことを確認する。ループ展開処理部（１０７）では実行回数を実行時に判定し分岐するコードとともに対応する展開部を生成する。ストレングスリダクション処理部（１０８）で展開後のループをさらに最適化する。

【効果】　コンパイル時に繰返し回数の決定できないループも展開でき、ループ展開の最適化効果を得ることができる。また展開されたループ本体に対してストレングスリダクションを実施することも可能になり、多くのＲＩＳＣ計算機にみられる制約されたアドレシングモードである場合にも演算を減少させる。

図１



1

【特許請求の範囲】

【請求項１】計算機言語で記述されたプログラムのソースコードから実行用のオブジェクトコードを生成するコンパイラにおいて、繰返し構造を表現しているループのプログラムソースをオブジェクトコードに変換する方法であって、

コンパイル時には該ループの繰返しの実行回数が不明であるが、実行時に該繰返し回数が、そのループの入口で確定するものに対して、該ループの入口において繰返し回数を実行時に判定し、それに応じて繰返すべき回数だけ、もしくはそれを複数の数の和として表し、それぞれ該複数の数だけループ本体の複製を作成することによって展開を実施することを特徴とするループ最適化方法。

【請求項２】請求項１記載のループ最適化方法において、該ループの入口における実行時の繰返し回数の判定を、その回数の偶数・奇数によって行い、それに応じて奇数回繰返しの場合と偶数回繰返しの場合とに分けて、繰返すべきループ本体の複製を作成し、展開することを特徴とするループ最適化方法。

【請求項３】計算機言語で記述されたプログラムにおいて繰返し構造を表現しているループに関して、その実行速度を向上させるためのプログラム変換手法を、請求項１記載の方法において展開された後のループの繰返し本体に適用することを特徴とするループ最適化方法。

【請求項４】計算機言語で記述されたプログラムにおいて繰返し構造を表現しているループに関して、その実行速度を向上させるプログラム変換手法の一つであるストレングスリダクション方式を、一つのループに対して複数回適用することを特徴とするループ最適化方法。

【請求項５】計算機言語で記述されたプログラムにおいて繰返し構造を表現しているループに関して、その実行速度を向上させるプログラム変換手法の一つであるストレングスリダクション方式を実施せずに請求項１記載の方法において、該ループを展開し、その結果できる展開後のループに対して新たにストレングスリダクションを適用するループ最適化方法。

【請求項６】請求項２に記載の条件で展開したループに対して、請求項４に記載のストレングスリダクション方式を実施することを特徴とするループ最適化方法。

【請求項７】請求項１に記載のループ展開方法において、該展開処理を実施するか否かを、展開によって生じるオブジェクトコードのサイズの増加量と、ユーザが指定するコンパイル時のオプションとの組合せによって決定することを特徴とするループ最適化方法。

【発明の詳細な説明】

【０００１】

【産業上の利用分野】本発明はコンピュータプログラムのコンパイラにおける最適化方式に係り、最適化コンパイラが、より効率的に実行されるようなコードを生成することを可能にする方法に関する。本発明はＲＩＳＣ

2

（Ｒｅｄｕｃｅｄ　Ｉｎｓｔｒｕｃｔｉｏｎ　Ｓｅｔ　Ｃｏｍｐｕｔｅｒ）に特に適応性を有しているが、一般のコンパイラにおいても利用可能である。

【０００２】

【従来の技術】コンパイラが生成するオブジェクトコードの実行時の効率を向上させる最適化技術が従来より種々提案されている。中でも、プログラム中で実行頻度が高く、実行時間の多くの割合を占める部分はループであるので、ループの最適化手法が特に重要である。これらの従来の最適化技法については、佐々正孝著「プログラミング言語処理系」（岩波講座ソフトウェア科学５、１９８９年）第９章４４１頁から４９９頁などに記載されている。

【０００３】このループ最適化手法の中で本発明に関するものとして、ループ本体を何回か複写して繰返しの終了判定の回数を削減するループ展開方法とループの繰返し本体内で実行される演算の削除／複雑度の軽減（より実行速度の速い演算による置き換え）操作を取り上げる。

【０００４】まず、ループの展開は、ループの繰返し数分だけ、もしくはその約数分だけその本体を複写して順に配置しすることである。例えば、次の１００回繰返しのループ

```
loop　１００回
  ループ本体
end
```

を本体を４回複写して並べ、２５回繰返しのループ

```
loop　２５回
  ループ本体
  ループ本体
  ループ本体
  ループ本体
end
```

のようにすることがループ展開の典型である。ループ展開すると、上記の例では展開前に１００回であった終了の判定を展開後には２５回に削減できるので、実行速度を向上することができる。従来、この展開操作はループの繰返し回数がコンパイル時に既知である場合にのみ利用可能であった。

【０００５】次に、演算の強度の軽減についてである。これはループの繰返しにともなって定数値ずつ更新されるような変数（これを帰納変数と呼ぶ）の一次式に対し、その定数倍の部分において実施される乗算を加算に置き換えることを主とするものである。例えば、次のループ

```
loop
  x　＝　a［i］
  i　＝　i＋1
end
```

において、ａ［ｉ］のアドレス計算では、通常、ｉ＊

「配列要素のサイズ（Ｌとする）」が生成される。ここで、ｉが帰納変数である。一般に乗算の計算コストは加算などに比べて大きいのでループ内でｉ＊Ｌを計算した後ｉ＝ｉを行うかわりに、ｉ－Ｌだけを行って繰返しを進行するものである。ｉが０から始まるものとすると、次のように新たな変数ｉ’を導入してこのループを変換する。

【０００６】

```
i'  =  0 (iの初期値)
loop
   x  =  [] (&a+i')
   i'  =  i'  +  L
end
```

ここで、［］は添字操作を行う演算子で、ここでは引数（＆ａ＋ｉ’）で示される値をアドレスとして、配列要素の参照を実現するものである。また、＆はその引数（ここではａ）のアドレスを得る操作であるものとする。実際の計算機の命令では、＆ａ＋ｉ’は＆ａを（ベースレジスタ、オフセット）で表現し、ｉ’をインデクスレジスタで表現することになるが、ＲＩＳＣ計算機ではベース、オフセット、インデクスという上記３種からなるアドレシングの命令をもたないことが多い。そこで、さらに次のようにして、変数ｉ”を導入してこれをベースレジスタの内容＋オフセットに設定し、

```
i"  =  &a+0
loop
   x  =  [] (i")
   i"  =  i"+L
end
```

の形式で変換する。

【０００７】これによりループ本体でコスト大の乗算が、よりコスト小の加算に置換され、実行速度が向上する。乗算に限らず、ループ内の加算などでは、それを削除することもでき、このような演算の強度（コストの大きさ）を軽減する変換操作はストレングスリダクションと呼ばれる。

【０００８】

【発明が解決しようとする課題】上記従来技術では、コンパイル時に実行回数が不明であるループは展開の対象としていなかった。さらに、ストレングスリダクションにおいてはこれを実施するか実施しないかの二者択一でありループ展開と組み合わせて、展開後のある部分には適用し、ある部分には適用しないというように、これらを組み合わせる最適化方式は適用されていなかった。

【０００９】さらに、ループ展開処理は一般にオブジェクトコードのサイズを増大させるので、ループ本体のサイズが一定値以下の場合にしか適用されず、実行速度とオブジェクトコードサイズの重要度に応じて柔軟に制御することができなかった。

【００１０】本発明の目的は、コンパイル時には実行回数不明のループにも展開操作を実施して実行速度を向上させ、この展開されたループに対してストレングスリダクションを実施し、ループ構成命令を使用する機会を増やして一層の速度向上を得るとともに、実行速度とコードサイズとの重要度に応じて展開可否の条件をユーザが指定できる高性能なループ処理方式を提供するものである。

【００１１】

【課題を解決するための手段】上記の目的を達成するために、コンパイラはコンパイル時に次の手順でループの展開、およびストレングスリダクションを実施する。

（０）ループ本体のソース言語または中間語でのサイズとコンパイルオプションでの指定から、ループを展開するべきであるか否か、展開する場合の最大展開数を決定する。

（１）繰返し回数不明のループに対しては、回数がループ内で変更を受けないことを確認した後、そのループの入口で回数を判定し、その値によってそれぞれ対応する展開部分へ分岐するコード（「判定コード」と呼ぶ）を出力する。

【００１２】（２）（１）での分岐のターゲットとなるよう、展開において複写したループ本体は、１回展開部、２回展開部、３回展開部、．．．のように、展開個数に応じて必要なだけ置く。（２－１）繰返し回数不明のループと展開数かループ繰返し数未満となるループでは、１回展開部、２回展開部、３回展開部、．．．とすべての展開部を置き、各展開部の最後にそれぞれ（１）の「判定コード」を置く。（２　２）繰返し回数かコンパイル時に既知で、かつ展開数か繰返し数以上の場合、繰返し数に等しい展開数の展開部だけを置く。例として、２回展開に限定する場合、（２－１）の条件に対応したものでは、１回展開部ではその最後で１回分の終了判定を置き、２回展開部ではその最後で終了判定してこの２回展開部自身を繰り返す形式となる。

（３）展開後のループに対して２回展開部では帰納変数＋１、３回展開部では帰納変数－１、帰納変数＋２．．．のそれぞれを対象としてストレンクスリタクションを実施する。

【００１３】

【作用】コンパイル時にループ繰返し回数か決定しているとき、実行時には、その繰返し回数もしくは最大の回数分に対応する展開後本体に分岐する。また、ループ入口の回数判定部分で実行時にループの繰返し回数が確定するときは、その回数を実行時にループ入口で判定して、対応する回数分の展開後本体に分岐する。展開後の本体では、上記最適化のための手段によってループの繰返し回数もしくはその因数分の実行文か配置されているので、終了判定無し、もしくは因数の個数分の判定回数で繰返しの実行を終了してループを脱出する。ここで繰返し回数の因数とは、それらの総和か繰返し回数に等し

くなるような整数という意味で使用している。

【００１４】展開後の本体内では、帰納変数＋１、帰納変数＋２、．．．に対して、それらを添字とする配列参照で、配列の起点＋それらをそれぞれベースアドレスとしてレジスタに保持しておき、ストレングスリダクションを実施し、ループ本体内での演算を軽減する。

【００１５】

【実施例】以下、図を用いて本発明の実施例を詳細に説明する。図１は本発明の一実施例を示すものである。コンパイラは図１にあるように、ソースコードの字句解析・構文解析（１０１）、意味解析・中間コート生成（１０２）、大域的最適化（１０３）と進行する。ここまでの処理およびコート生成（１０４）での処理は従来からのものである。

【００１６】本発明は、大域的最適化（１０３）に改良を加えることであり、まずコンパイル時に繰返し回数が不明てあるループを２回展開することを例にとり、図２のプログラム片を用いて説明する。図２のループは繰返しを制御する変数ｉが０からＮまで１ずつの刻みで動き、ｉを添字として配列ａの要素を順次参照するものてある。Ｎの値はコンパイル時には既知でないものとする。コンパイル時の処理は以下のとおりである。

【００１７】図１において、１０５のループ認識部で認識されたループ（図２　２０１）に対し、実行回数確定処理（１０６）を実施する。それに先だって展開条件検査部（１０７）において、このループを展開するべきかとうかを決定する。これは、今対象としているループの本体のサイス（大域的最適化か処理対象とするプログラムの中間表現でのサイス）か、あらかじめ設定されている値を超えない時に展開可能とするものてある。この設定値はユーサによって変更することかてき、コンパイルオプションによってより大きなもの、または制限なし等に変更できる。

【００１８】以降の例では、ループか展開可能てあるものとする。１０７では、対象としているループの実行回数がその入口点で決定され、ループ本体の実行時には変更を受けないことを検査する。ある種のプログラミング言語では言語仕様上で上記の条件か保証されているか、Ｃ言語などではそれが保証されておらず、コンパイラによる検査か必要である。この検査の手法は従来より実施されている最適化処理てのテータフロー解析を利用して行う。図２のループでは、繰返しの上限値を表現している変数Ｎがループの本体内で設定されないことを確認することでなされる。

【００１９】繰返し回数か確定すると、本ループか展開対象てあるとされ、図１のループ展開部（１０８）て展開処理される。図３にその展開処理の詳細を示し、図４に展開結果のコードを示す。図３の処理を行うための確認処理の手順は以下の通りてある：

「１」帰納変数の検出

ループ中でｉ＝ｉ＋Ｃ１（Ｃ１は定数）という形式で一度だけ定義されている変数（これを帰納変数と呼ぶ）を見つける。

「２」帰納変数の初期値の検出

「１」で見つけた帰納変数のループ入口での値を求める。これを帰納変数の初期値と呼び、以下ではＩ０で表す。

「３」ループ終了判定用の比較式の識別

ループを終了する際の判定用の比較式の形式か「１」で見つけた帰納変数ｉを用いてｉ＜Ｃ２またはｉ＜＝Ｃ２（ここでＣ２は定数またはループ中で値の変わらない変数）となっていることを確認する。

【００２０】「４」繰返し回数の認識

Ｉ０，Ｃ１、Ｃ２の組合せによって、繰返し回数がコンパイル時に既知となるか否か、および既知だとしたら偶数回か奇数回かか以下に示すように分かる。

場合１：＜Ｉ０定数、Ｃ２定数＞

繰返し回数はコンパイル時に既知てあり、（Ｃ２－Ｉ０）／Ｃ１（判定用の比較式の種別が＜の場合）あるいは（Ｃ２－Ｉ０－１）／Ｃ１（判定用の比較式の種別が＜＝の場合）である。

場合２：＜Ｉ０定数、Ｃ２定数てない＞

場合３：＜Ｉ０定数でない、Ｃ２定数＞

場合４：＜Ｉ０定数でない、Ｃ２定数てない＞

これらの場合、繰返し回数はコンパイル時に既知でない。

【００２１】「５」展開処理の実施（図３における処理）

回数判定コート生成部（３０１）および分岐コード生成部（３０２）において、上記手順によって得られた繰返し回数にしたがいそれぞれ対応する展開本体へ分岐するコート（図４　４０１、４０２）を生成する。さらに、ループ本体のサイスとコンパイル時のオプションから展開の回数を決定する。本例では２回の展開を実施し、繰返し回数の偶数／奇数を判定して分岐するものとする。

【００２２】次に３０３～３０６によって本体の展開部分を配置する。本例では、ループ本体部分は１回展開部（本体そのまま（図４　４０３））と２回展開部（図４　４０６）とから構成される。展開された本体部では、展開数だけ帰納変数を更新する命令を配置する。３０６ての処理により１回展開部の最後は、本ループが１回たけ繰返して終了する場合のための終了判定（図４　４０４，４０５）てある。２回展開部の最後てはこの２回展開部自身を繰返すかとうかの判定コート（図４４０７、４０８）を配置する。３０４，３０５，３０６の後処理てはこのように各展開ての終了判定や新たな繰返しへの判定コートを生成する。なお、この配置は一例であり、奇数回のときに２回展開部の中間地点に飛び込むような配置も考えられる。

【００２３】ここまてて、本例て出力されたオブジェク

トコードを実行した場合の実行過程を図４によって追うと、ループの入口において、

（ａ）繰返し回数が奇数回のとき

４０１、４０２の判定、分岐によって、奇数の場合次に配置された１回展開部（４０３）を実行する。繰返しが１回のときは１回展開部の最後の判定（４０４、４０５）でループを脱出しラベル　Ｌ@ｅｘｉｔ（４０９）に分岐する。３回以上のときはこの判定（４０４、４０５）で分岐せずに２回展開部を実行する。２回展開部では帰納変数ｉを添字とした配列参照をｉとｉ＋１の二通りで行い、ｉ自身は２だけ更新するようにする。これに関しては以下のストレンクスリタクションの節で述べる。このｉの処理ついて、ｉでの参照、１だけ更新を２回繰り返す方式もある。２回展開部（４０６）の最後の判定部（４０７，４０８）で帰納変数が繰返しの上限値を超えたか否かの判定を実施してループを脱出する／繰返すの制御を行う。ループを繰り返す場合はこの２回展開部自身の先頭へ分岐するようラベル　Ｌ@ｅｖｅｎを参照する。

【００２４】（ｂ）繰返し回数が偶数のとき

４０１，４０２によって２回展開部に分岐し、以下（ａ）の３回以上の場合と同様の動作を行う。このループての具体的な命令語系列は図５の通りとなり、５００がベースアドレスやカウンタの設定である前処理部で、４０１－５０１、４０２－５０２、４０３－５０３、４０４－５０４、４０５－５０５、４０６－５０６、４０７－５０７、４０８－５０８、のように図４と図５とが対応することになる。

【００２５】次に、上記の例のコンパイル処理で展開されたループに対して、さらに最適化を行うために、コンパイル時の処理として図１　１０９でストレングスリタクション処理を実施することについて図５を用いて説明する。ここて、配列は添字値０から始まり、配列要素のサイズは４であるものとする。また、対象となる計算機の命令語にはインデクスモートによるアドレシングは、「ベースレジスタの内容＋インデクスレシスタの内容」を実効アドレスとするものだけしかないとする。

【００２６】２回展開を実施すると、ソースコードのイメーシでは、図２のフロクラムにおいては以下のように変換され

```
loop  i  =  0..N
   x  =  a[i]
   i  =  i+1
   x' =  a[i]
   i  =  i+1
end
```

となり、ａ［ｉ］の参照とｉを－１更新した後のａ［ｉ］の参照が生成される。ｘ'は展開のために生したｘの複写てある。ここて後者のａ［ｉ］の参照は、ｉを－１更新しないてａ［ｉ－１］の参照とすることかてき、次のようにａ［ｉ＋１］が２回展開部に生成されることになる。

【００２７】

```
loop  i  =  0..N
   x  =  a[i]      ----(1)
   x' =  a[i+1]    ----(2)
   i  =  i+2
end
```

このうち前者のａ［ｉ］の参照（１）はｉをインデクスレジスタに保持して、ｒ１をベースとするインデクスモードでの参照（５０６の最初のＬｏａｄｘ）で行う。ａ［ｉ＋１］の参照（２）に関しては、これを（ａ＋１）［ｉ］と考えて、ｒ２をベースとしｉをインデクスとするインデクスモートでの参照（５０６の２番目のＬｏａｄｘ）で行う。ここで（ａ－１）は配列ａの先頭から１要素分離れたアドレスを示すものとして考える。したがって、配列の起点アドレスとしてａ［０］およびａ［１］のアドレスをベースとしてレジスタｒ１，ｒ２に保持するようにする（それぞれ５００の２つのＬｏａｄａ）。５００の２番目のＬｏａｄａでは配列要素のサイズが４てあるので、ｒ１から４隔たった値としてｒ２を設定するようにしている。また、Ｌｏａｄｃはインデクスレジスタの初期値設定、Ｓｈｆｔｌは終了値を検査するために繰返し回数（ｒＮ）の４倍を設定するものである。これらの操作は前処理として当該ループの入口で繰返し前に実施する。

【００２８】以上の処理により、５０６において、２つめのＬｏａｄｘか元のａ［ｉ］の参照に対応するもののままであれば、ｉ＝ｉ＋１がその直前にあるへきであるか、ここてａ［ｉ－１］に対応するものとしたので、このｉ＝ｉ＋１を削除することかてきる。ここでの操作は、元々は２つのＬｏａｄｘに対する２つのｉ－１が生成されるところを、そのうちの１つを省略して１つのｉ＋２にし、２に対して配列要素のサイス４による乗算を＋８の更新（５０６てのＡｄｄｃ）としたストレンクスリタクションである。

【００２９】図５が最終的に変換されたループの命令系列である。ここで、ＳＰはヘースレシスタてあり、各命令はそれぞれ次のような意味をもつものとする。

Ｌｏａｄａ　ｒ，ｄ（ＳＰ）…ＳＰからｄ（定数）だけ隔たった地点のアトレス自身をレシスタｒにロートする

Ｌｏａｄｃ　ｒ，ｉ　…定数ｉをレシスタｒにロードする

Ｌｏａｄｘ　ｒ，ｘ［ｂ］　…レシスタｂをヘース、レジスタｘをインデクスとしてアドレスを得、そこに格納されている値をレジスタｒにロートする

Ｓｈｆｔｌ　ｒ，ｉ，ｔ　…レシスタｒの値を左へｉヒットシフトし、レシスタｔに格納する

Ａｄｄｃ　ｒ，ｉ　…レシスタｒに定数ｉを

9

加える

Ｔｅｓｔ　　ｒ　　…レジスタｒの値を検査する

Ｃｏｍｐｃ　　ｒ，ｉ　　…レジスタｒの内容と定数ｉを比較する

Ｃｏｍｐ　　ｒ，ｔ　　…レジスタｒの内容とレジスタｔの内容を比較する

Ｂｃｏｎｄ　　ｃ，ｌ　　…比較、検査結果がｃの条件であるときラヘルｌに分岐する

次に、３回に展開する場合を例として考える。このとき繰返し回数か定数で３回の場合、コンハイル時に３０１でそのことか判明するので、３０２においては3回分の展開部たけを生成する。繰返し回数に応じた分岐コードや終了判定コートは生成しない。

【００３０】繰返し回数かコンハイル時には既知でない場合を、図６を用いて説明する。コンパイル時に図３　３０３の処理で１回、２回、３回の展開部を置く（それぞれ６０２、６０４、６０６）。３０２ではそのそれぞれに対応して、繰返し回数を判定して１回のとき、２回のとき、３回以上のときにそれぞれ対応する展開部へ分＊岐するコードを配置する。この分岐コードを図６　６０１に示す。６０１はｌと比較して等しければ分岐、２と比較して等しければ分岐、３と比較して以上であれば分岐という形式であるが、これ以外にインデクス分岐命令を使用する形式もありえる。最大展開数である３以外の各展開部の最後ではこれで繰返しが終了するので、ループを脱出するコード（６０３、６０５）を配置する。これは図３３０４、３０６での処理で実現されるものである。3回展開部（６０６）では、インデクスｒｃの更新に加え、繰返し回数ｒＮを－３ずつ更新する。６０６の最後では、新たな繰返しのために、６０１の判定部を実行するようなコードを配置する。６０１に戻るよう分岐してもよいし、その判定部を新たに配置してもよい。ここでは新たに６０７を配置する形式とする。変換されたループの模式図は図６のようになる。

【００３１】ここで、実行時に繰返し数が１７回であったとすれば、５回展開部を３回、２回展開部を１回それぞれ実行してループを終了する。この例て導入した新たな命令は以下である。

10

Ｂ　　ｌ　　…ラヘルｌに無条件分岐する

さらに、別の例として展開部を２のべき乗のものだけ用意して、繰返し回数の二進表現の各ビットが０であるか１であるかによって終了判定／分岐を行う例に関して図７を用いて述へる。はじめに説明した偶数／奇数でわける２回展開の例は、これの特別な場合てあるとも考えられる。ここて、繰返し回数はコンハイル時に既知でないものとする。

【００３２】コンハイル時に展開部は本体自身（２の０乗回）と２回（２の１乗回）展開、４回（２の２乗回）展開を用意する。３０３の処理て本体（７０２）、２回展開部（７０３）、４回展開部（７０４）を配置する。３０２では、繰返し回数の最下位ヒットから順に‘１’であるかを判定し、‘１’であれは対応する展開部へ分岐するコード（７０１）を配置する。ただし、最終の展開部である４回展開部では繰返し回数か４以上を判定する（２９ヒット目以降の上位ヒットが‘１’という判定でもよい）。

【００３３】最終以外の各展開部の終了判定では、図３　３０６の処理として、自身に対応するヒット位置以下は、実行時にここに来た段階てはすでに繰返し終了済となるので、コンパイル時の処理として、自身の次のヒット位置を判定する部分へ戻るようにする（７０２、７０３でのＢ命令）。最終てある４回展開部の最後では４以上てあるかの判定を行って自分自身を繰返すように命令を配置する（図３　３０５）。

【００３４】

【発明の効果】本発明によれは、コンハイル時に繰返し回数の決定できないループも展開てき、ループ展開での最適化効果を得ることが可能になる。また展開されたループ本体に対して、帰納変数ｉ、ｉ＋１、ｉ＋２、．．．を対象にストレングスリダクションを実施することでｉの加算を軽減することも可能になる。このストレングスリタクションは多くのＲＩＳＣ計算機にみられるように、ベース＋インデクス＋オフセットでのアドレシングモードをもたず、ベース＋インテクスて対処すへき場合にも演算を減少させることかできるものてある。

【図面の簡単な説明】

【図１】本発明の最適化操作を実施するコンハイラの構成図である。

【図２】本発明の適用例となるフロクラムである。

【図３】本発明の最適化操作を示す手順の図てある。

【図４】本発明の一実施例によって展開されたループの構造を示す図てある。

【図５】本発明を適用したコンパイラの出力コートの例てある。

【図６】本発明を適用したコンハイラの出力コートの別の例である。

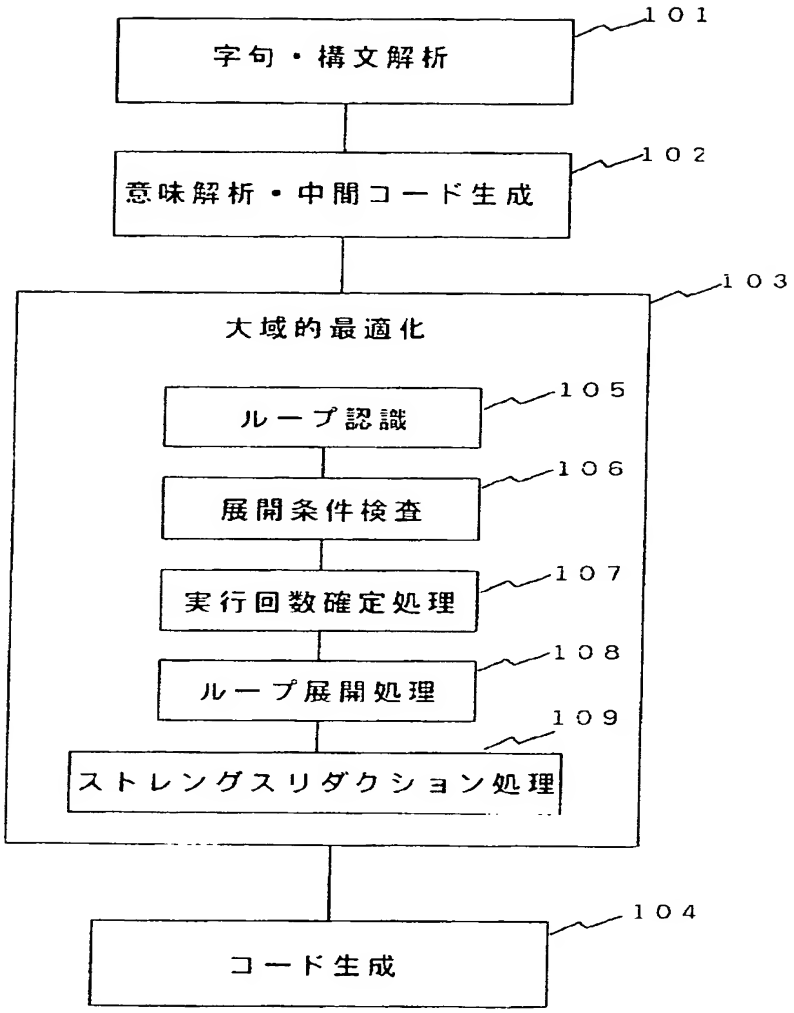
【図７】本発明を適用したコンハイラの出力コートの別の例である。

【符号の説明】

１０３…大域的最適化部、１０５…ループ認識部、１０６…実行回数確定処理部、１０７…ループ展開処理部、１０８…ストレングスリタクション処理部。

【図1】

図1



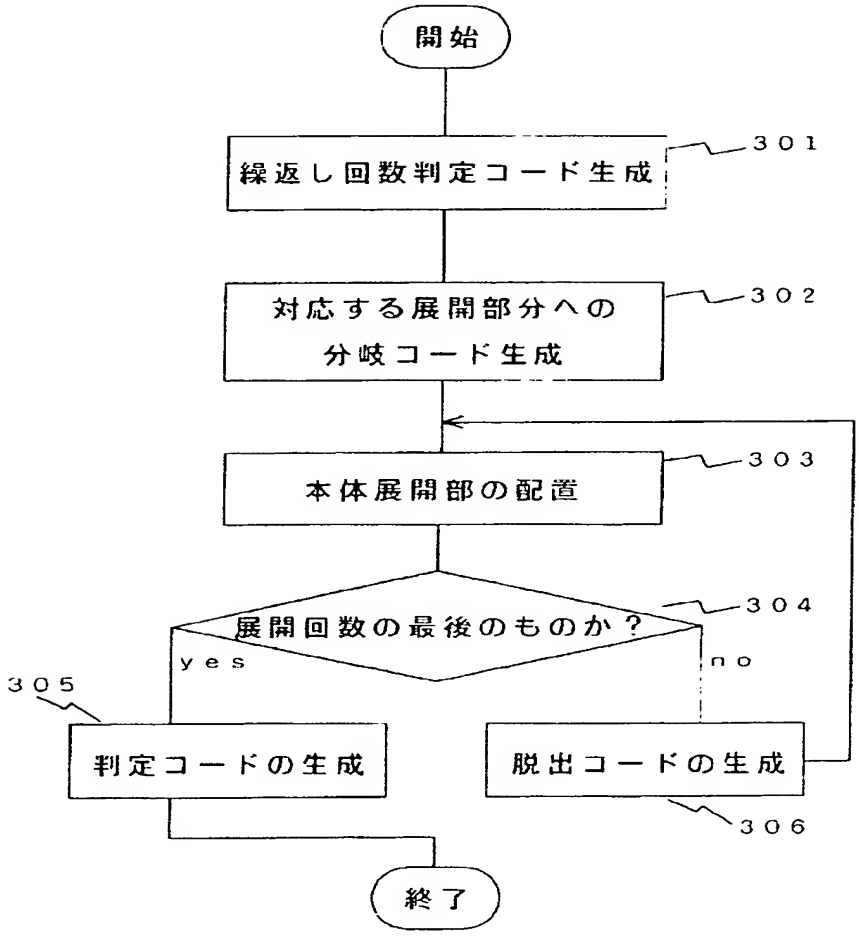
【図2】

図2

```
loop  i  =  0__N  by  1
    x  =  a[i]
end
```

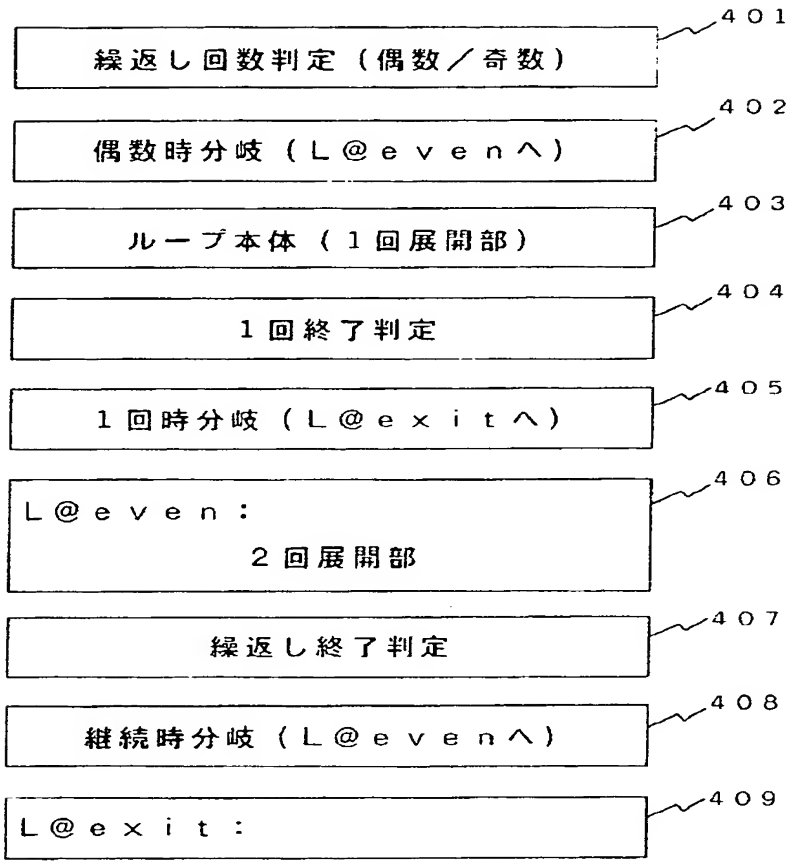
【図３】

図３


開始
繰返し回数判定コード生成
３０１
対応する展開部分への
分岐コード生成
３０２
本体展開部の配置
３０３
展開回数の最後のものか？
３０４
yes
no
３０５
判定コードの生成
脱出コードの生成
３０６
終了

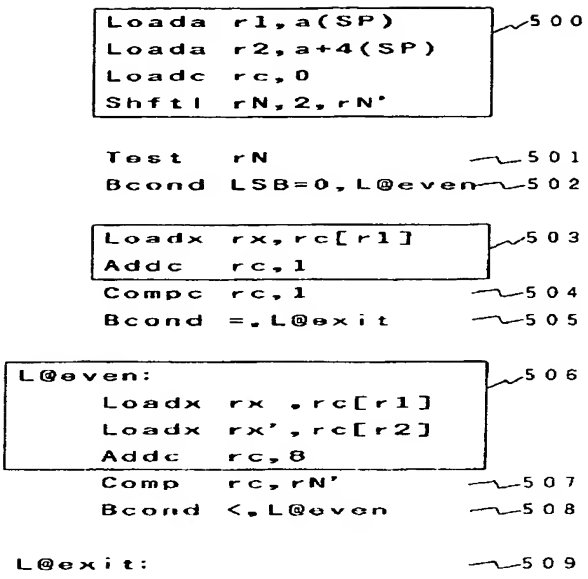
【図４】

図４


繰返し回数判定（偶数／奇数）
401
偶数時分岐（Ｌ＠ｅｖｅｎへ）
402
ループ本体（１回展開部）
403
１回終了判定
404
１回時分岐（Ｌ＠ｅｘｉｔへ）
405
Ｌ＠ｅｖｅｎ：
２回展開部
406
繰返し終了判定
407
継続時分岐（Ｌ＠ｅｖｅｎへ）
408
Ｌ＠ｅｘｉｔ：
409

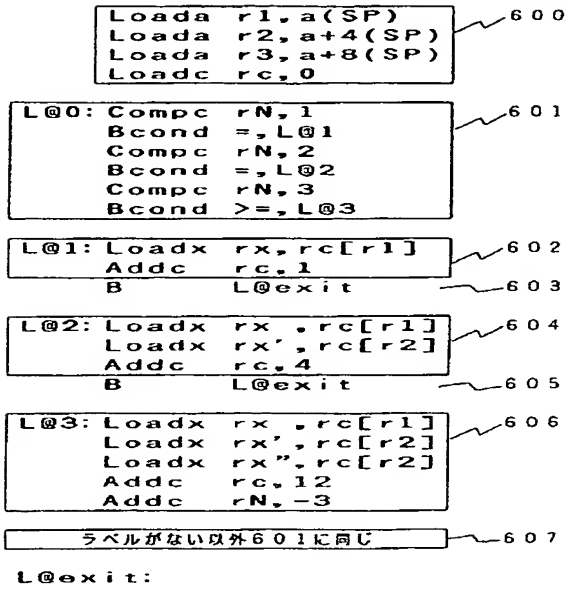
【図5】

図5



【図6】

図6



【図7】

図7